Chapter 2

Minimization/Optimization – Problems

Literature

- W. PRESS *et al.*, *Numerical recipes*, Cambridge university press
- F.S. ACTON, *Numerical Methods that work*, Mathematical Association of America

Optimization is of enormous importance in physics, engineering, economics, information technology and in many other fields. The layout of electric circuits on a chip, timetables, the optimum load of a pipeline system are a few, typical examples.

In general we have a problem which is determined by n parameters and one has to find the specific values of those parameters for which a so-called *cost function* develops a maximum/minimum. This an be expressed in a more formal way:

> There is a finite or infinite number of states. Each state is defined by a set of n real parameters $\{p_i \,|\, i = 1, \ldots, n\}$. The set of all states spans the *search* space $\mathbb{S}$. Each state is correlated to a real number, its *cost*. The function generated this way is the *cost function* $F(\ldots)$. Thus, if $F(\ldots)$ is this cost function, $\mathbb{S}$ the search space, and $\mathbb{R}$ the set of real numbers, we have:
>
> $$F \;:\; \mathbb{S} \;\longrightarrow\; \mathbb{R}.$$
>
> It is the purpose of an optimization algorithm to find the state for which the cost function develops a (global) maximum/minimum.

Is the cost function at least once differentiable in its parameters then mathematics will provide us with well defined (deterministic) algorithms to find the maximum/minimum of the cost function. Such methods will be discussed in Secs. 2.1 to 2.3.

In all other cases methods which have been developed from evolutionary models come to play. Most of the time one starts with some state $\mathbf{x} \in \mathbb{S}$ and modifies this state using *stochastic methods*. Does the cost function result in a bigger/smaller value the new state is accepted and serves as a basis for the next modification, otherwise the old state is modified again. This is repeated until the maximum/minimum has been found. This is the most simple form of a *Mutation-Selection Algorithm*. More sophisticated methods will be discussed in Secs. 2.4 to 2.8.

2.1 Continuous Degrees of Freedom

We will discuss problems with many continuous degrees of freedom for which standard methods fail. Some representative examples are

1. The **ground state energy** E_0 of a quantum mechanical system is given by the minimum of the energy expectation value

$$E_0 = \min_{\psi} \frac{\left\langle \psi \middle| \hat{H} \middle| \psi \right\rangle}{\langle \psi \,|\, \psi \rangle}. \tag{2.1}$$

   Expressed in a complete orthonormal basis set $\{|\Phi_i\rangle\}$ this is equivalent to

$$E_0 = \min_{\mathbf{c}} \frac{\mathbf{c}^\dagger \boldsymbol{H} \mathbf{c}}{\mathbf{c}^\dagger \mathbf{c}}, \tag{2.2}$$

   where $\mathbf{c}$ stands for the vector of expansion coefficients of the ground state vector and $\boldsymbol{H}$ is the Hamilton matrix in the basis $\{|\Phi_i\rangle\}$, *i.e.:*

$$H_{ij} = \left\langle \Phi_i \middle| \hat{H} \middle| \Phi_j \right\rangle.$$

2. Variational ansatz with parameters $\boldsymbol{\eta}$

$$|\psi\rangle = |\psi(\boldsymbol{\eta})\rangle, \quad \boldsymbol{\eta} \in \mathbb{R}^n \text{ parameters}. \tag{2.3}$$

   Again the energy expectation value is to be minimized:

$$E_0 = \min_{\boldsymbol{\eta}} \frac{\left\langle \psi(\boldsymbol{\eta}) \middle| \hat{H} \middle| \psi(\boldsymbol{\eta}) \right\rangle}{\langle \psi(\boldsymbol{\eta}) \,|\, \psi(\boldsymbol{\eta}) \rangle}. \tag{2.4}$$

3. The solution of *a set of linear algebraic equations* can be viewed as minimization problem

$$
\begin{aligned}
\boldsymbol{B}\,\mathbf{x} = \mathbf{b} \quad \Longleftrightarrow \quad & \min_{\mathbf{x}} \|\boldsymbol{B}\,\mathbf{x} - \mathbf{b}\|^2 \\
& = \min_{\mathbf{x}} \left[\mathbf{x}^\dagger B^\dagger B\mathbf{x} - 2\mathbf{b}^\dagger \boldsymbol{B}\mathbf{x} + \|\mathbf{x}\|^2\right] . \qquad (2.5)
\end{aligned}
$$

   For large-dimensional matrices $\boldsymbol{B}$, the direct inversion may easily exceed present computer power. We will see that iterative minimization procedures come in handy also for this problem.

4. In quantum molecular dynamics one is interested in the dynamical properties of molecules or solids. In an entirely classical approach, only the positions of the nuclei, the centers of the atoms under consideration, are accounted for and empirical forces among the nuclei are introduced, which effectively mimic parts of the quantum mechanical features. In general only effective two-particle forces are employed, which, for the i-th particle, are of the form

$$
\mathbf{F}_i = \sum_j \mathbf{f}(\mathbf{R}_i, \mathbf{R}_j),
$$

   with $\mathbf{R}_i$ being the coordinate of the i-th particle. The classical equations of motion for the nuclei read

$$
m_i \ddot{\mathbf{R}}_i = \mathbf{f}(\{\mathbf{R}_i\}).
$$

   A first step towards complete quantum molecular dynamics consists in treating at least the forces quantum mechanically by solving the electron dynamics quantum mechanically while the nuclei are treated as classical degrees of freedom. This approach is a significant improvement and goes beyond the classical approximation and is known under *Born-Oppenheimer approximation*. It can be justified by the observation that the mass of the nuclei is roughly 2000 times heavier than that of the electrons. The dynamics is then solved in two steps. This way one will be able to assess the quality of the Born-Oppenheimer approximation.

   The first step consists in solving the electronic eigenvalue problem for fixed positions $\{\mathbf{R}_i\}$ of the nuclei. This is a many-electron problem which, though in a fixed potential, is not soluble exactly. A widespread approximation for such type of problems is the local-density approximation (LDA) (the effective one-electron problem) for which the Kohn-Sham equations

$$
\begin{aligned}
\hat{H}(\{n\}, \mathbf{R})\, \psi_{\mathbf{k}}(\mathbf{x}) &= \varepsilon_{\mathbf{k}}\, \psi_{\mathbf{k}}(\mathbf{x}) \qquad \text{(Kohn – Sham)} \\
n(\mathbf{x}) &= \sum_{\mathbf{k} \le \mathbf{k}_{Fermi}} |\psi_{\mathbf{k}}(\mathbf{x})|^2 \qquad (2.6)
\end{aligned}
$$

have to be solved self-consistently via the electronic density $n(\mathbf{x})$. The reasoning is that for low temperatures and fixed nuclei the electronic subsystem is in the ground-state. Since the motion of the nuclei is slow compared to that of the electrons, the dynamics of the nuclei is equivalent to an adiabatic change of the potential which leaves the electronic subsystem always in the eigenstate (ground state) which corresponds to the actual potential. The ground state energy $E(\{\mathbf{R}_j\})$, consisting of contributions from the electrons and the nuclei, defines the potential in which the nuclei move.

The second step consists in computing the force on the i-th particle via the Hellmann-Feynman formula

$$\mathbf{F}_i = -\frac{\partial}{\partial \mathbf{R}_i}\, E(\{\mathbf{R}_j\}) = -\left\langle \psi \,\middle|\, \frac{\partial}{\partial \mathbf{R}_i}\,\hat{H} \,\middle|\, \psi \right\rangle. \tag{2.7}$$

The change in the position of the nuclei for a small time-step is computed via the classical equations of motion, driven by the ab-initio forces. In hindsight we can scrutinize the reliability of the classical approximation for the nuclei. We can separate one nucleus and make a quadratic approximation to the ab-initio potential, leading to a harmonic oscillator for which the exact quantum mechanical dynamics can easily be obtained.

Molecular dynamics will be discussed in a later chapter. But the Car-Parrinello method also contains aspects of minimization problems. Often one is not interested in the vibrational modes but rather in the equilibrium geometry. Thus, we need to minimize the ab-initio potential

$$\min_{\{\mathbf{R}_j\}} \, E(\{\mathbf{R}_j\}),$$

which is obviously a challenging task as it involves the self-consistent solution of the electronic eigenvalue problem for each configuration $\{\mathbf{R}_j\}$. According to (2.4), the computation of the electronic ground-state is itself a minimization problem. We can therefore cast the entire problem into the form

$$\min_{\{\mathbf{R}_i\},\mathbf{c}} \left[\frac{\mathbf{c}^\dagger \boldsymbol{H}(\mathbf{c})\mathbf{c}}{\mathbf{c}^\dagger \mathbf{c}} + \sum_{i>j} V(\mathbf{R}_i, \mathbf{R}_f) \right].$$

Contrary to (2.4), however, the Hamiltonian matrix $\boldsymbol{H}$ depends itself on the expansion coefficients $\mathbf{c}$ because of its dependence on the density $n(\mathbf{x})$. The last term in the above equation describes the bare Coulomb interaction of the nuclei.

2.2 General Considerations on Quadratic Problems

A *quadratic form* is a scalar, quadratic function of a vector and is of the form:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \boldsymbol{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} + c. \tag{2.8}$$

[See also Eq. (2.5).] Here, $\boldsymbol{A}$ is a $N \times N$ matrix, $\mathbf{x}$ and $\mathbf{b}$ are vectors $\in \mathbb{R}^N$, and c is a scalar constant. We restrict the discussion to real-valued problems, and we want to demonstrate that if the matrix $\boldsymbol{A}$ is symmetric (i.e.: $\boldsymbol{A}^T = \boldsymbol{A}$) and positive definite (i.e.: $\mathbf{x}^T\boldsymbol{A}\mathbf{x} > 0$ for any non-zero vector $\mathbf{x} \in \mathbb{R}^N$) the quadratic form (2.8) is minimized by:

$$\boldsymbol{A}\mathbf{x} = \mathbf{b}. \tag{2.9a}$$

To prove this we set the gradient of $f(\mathbf{x})$ equal to zero:

$$\begin{aligned}
\nabla f(\mathbf{x}) &= 0 \\
&= \frac{1}{2}\nabla\left(\mathbf{x}^T\boldsymbol{A}\mathbf{x}\right) - \nabla\left(\mathbf{b}^T\mathbf{x}\right) \\
&= \frac{1}{2}\boldsymbol{A}^T\mathbf{x} + \frac{1}{2}\boldsymbol{A}\mathbf{x} - \mathbf{b} \\
&= \boldsymbol{A}\mathbf{x} - \mathbf{b}.
\end{aligned} \tag{2.9b}$$

If $\boldsymbol{A}$ is not symmetric, i.e.: $\boldsymbol{A}^T \neq \boldsymbol{A}$, then Eq. (2.9b) hints that one will find a solution to the system $\tilde{\boldsymbol{A}}\mathbf{x} = \mathbf{b}$ with $\tilde{\boldsymbol{A}} = \frac{1}{2}\left(\boldsymbol{A}^T + \boldsymbol{A}\right)$ which is a symmetric matrix.

It is now also possible to prove why the matrix $\boldsymbol{A}$ has to be positive definite for Eq. (2.9a) to work. Let us assume that $\mathbf{x}$ is a point $\in \mathbb{R}^N$ which satisfies $\boldsymbol{A}\mathbf{x} = \mathbf{b}$ and minimizes the quadratic form $f(\mathbf{x})$. Let $\mathbf{e}$ be an error term:

$$\begin{aligned}
f(\mathbf{x}+\mathbf{e}) &= \frac{1}{2}(\mathbf{x}+\mathbf{e})^T\boldsymbol{A}(\mathbf{x}+\mathbf{e}) - \mathbf{b}^T(\mathbf{x}+\mathbf{e}) + c \\
&\overset{(\boldsymbol{A}^T=\boldsymbol{A})}{=} \frac{1}{2}\mathbf{x}^T\boldsymbol{A}\mathbf{x} + \mathbf{e}^T\underbrace{\boldsymbol{A}\mathbf{x}}_{=\mathbf{b}} + \frac{1}{2}\mathbf{e}^T\boldsymbol{A}\mathbf{e} - \mathbf{b}^T\mathbf{x} - \mathbf{b}^T\mathbf{e} + c \\
&= \frac{1}{2}\mathbf{x}^T\boldsymbol{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} + c - \underbrace{\mathbf{e}^T\mathbf{b}}_{=\mathbf{b}^T\mathbf{e}} + \frac{1}{2}\mathbf{e}^T\boldsymbol{A}\mathbf{e} - \mathbf{b}^T\mathbf{e} \\
&= f(\mathbf{x}) + \frac{1}{2}\mathbf{e}^T\boldsymbol{A}\mathbf{e}.
\end{aligned}$$

If the matrix $\boldsymbol{A}$ is positive definite, then the second term is positive for all $\mathbf{e} \neq \mathbf{0} \in \mathbb{R}^N$ and, therefore, $\mathbf{x}$ minimizes $f(\mathbf{x})$.

The number of degrees of freedom, i.e.: the number of elements of $\boldsymbol{A}$ and $\mathbf{b}$, is of order N^2. If one of theses elements is modified the solution

$\mathbf{x}$ will change. This implies that the amount of information contained in the solution is also of order N^2. The gradient $\nabla f(\mathbf{x})$ has N components of information. A good method based on gradients should therefore need at most N iterations. The computational effort to compute the gradient N times is $\mathcal{O}(N^3)$, which is equivalent to the effort to compute the minimum via

$$\nabla f(\mathbf{x}) = \boldsymbol{A}\mathbf{x} - \mathbf{b} = 0 \Rightarrow \mathbf{x} = \boldsymbol{A}^{-1}\mathbf{b} \tag{2.10}$$

directly. At first glance, the two approaches seem to be equivalent as far as CPU time is concerned. This is not true, though, since iterative methods, based on gradients, have several advantages

- For sparse matrices $\boldsymbol{A}$ the effort to compute the gradient is proportional to the number of non-zero elements (typically $\mathcal{O}(nN)$, with $n << N$); i.e.: the entire approach is of order $\mathcal{O}(nN^2)$ instead of $\mathcal{O}(N^3)$.
- There is no need to really perform all N iterations. In most problems a much smaller number of iterations suffices for a desired accuracy.
- It can do with any amount of memory. There are three possibilities
  - keep the entire matrix in fast memory
  - read it from hard disk in portions, which can be kept in memory
  - generate the matrix elements from scratch as they are needed
- Optimally suited for parallelization or vectorization.

Method of Steepest Descent

The most elementary gradient-based method is the well-known steepest descent. The outline of the algorithm is as follows

Algorithm 5 Steepest descent (draft)

Choose a suitable initial vector $\mathbf{x}_0$
for $n = 0$ to $n_{\max}$ **do**
  calculate gradient $\mathbf{g}_n = \nabla f(\mathbf{x})\big|_{\mathbf{x}_n}$
  new search direction $\mathbf{r}_n = -\mathbf{g}_n$
  set $\mathbf{x}_{n+1}$ to the line minimum of $f(\mathbf{x})$ in direction $\mathbf{r}_n$
  if converged **then** EXIT
end for

When we take a step n, we choose the direction in which $f(\mathbf{x}_n)$ decreases most quickly, and this is the direction opposite $\nabla f(\mathbf{x}_n) = \boldsymbol{A}\mathbf{x} - \mathbf{b}$.

Suppose we start at the point $\mathbf{x}_0$. Our first step, along the direction of steepest descent, will fall somewhere on one of the solid lines in Fig. 2.1. Thus, we will choose a point

$$\mathbf{x}_1 = \mathbf{x}_0 + \lambda_0 \mathbf{r}_0, \quad \mathbf{r}_0 = \mathbf{b} - \boldsymbol{A}\mathbf{x}_0 = -\nabla f(\mathbf{x}_0). \tag{2.11}$$

The question is, how big a step we should take?

A *line search* is a procedure that chooses λ to minimize $f(\mathbf{x})$ along a line. This is the case when the *directional derivative* $df(\mathbf{x})/d\lambda$ is equal to zero. By the chain rule

$$\frac{d}{d\lambda} f(\mathbf{x}_n) = \nabla f(\mathbf{x}_n)^T \frac{d}{d\lambda}\mathbf{x}_n \overset{(2.11)}{=} \nabla f(\mathbf{x}_n)^T \mathbf{r}_{n-1}.$$

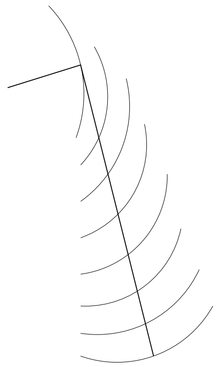


Figure 2.1: Line minimum.

Setting this expression equal to zero, we find that λ should be chosen so that $\mathbf{r}_0$ (the *residual*) and $\nabla f(\mathbf{x}_1)$ are orthogonal. As we always go to the respective line minimum, successive gradients are perpendicular. This is illustrated in the Fig. 2.1 which shows two successive search directions (gradients) along with the contours of $f(\mathbf{x})$. Obviously, if the gradients were not orthogonal, we could still reduce $f(\mathbf{x})$ by moving further in the direction of the old gradient. In mathematical terms the orthogonality of successive gradients follows from the line minimum condition.

To determine λ we proceed with:

$$\begin{aligned}
0 &= \mathbf{r}_1^T \mathbf{r}_0 \\
&= (\mathbf{b} - \boldsymbol{A}\mathbf{x}_1)^T \mathbf{r}_0 \\
&= [\mathbf{b} - \boldsymbol{A}(\mathbf{x}_0 + \lambda_0 \mathbf{r}_0)]^T \mathbf{r}_0 \\
&= (\mathbf{b} - \boldsymbol{A}\mathbf{x}_0)^T \mathbf{r}_0 - \lambda_0 (\boldsymbol{A}\mathbf{r}_0)^T \mathbf{r}_0 \\
&= \underbrace{(\mathbf{b} - \boldsymbol{A}\mathbf{x}_0)^T}_{=\mathbf{r}_0^T} \mathbf{r_0} - \lambda_0 \, \mathbf{r}_0^T \boldsymbol{A}\mathbf{r}_0 \\
\lambda_0 &= \frac{\mathbf{r}_0^T \mathbf{r}_0}{\mathbf{r}_0^T \boldsymbol{A}\mathbf{r}_0}.
\end{aligned}$$

Another reason why the matrix $\boldsymbol{A}$ is to be positive definite! This result can easily be generalized to step n and yields

$$\lambda_n = \frac{\mathbf{r}_n^T \mathbf{r}_n}{\mathbf{r}_n^T \boldsymbol{A}\mathbf{r}_n}, \tag{2.12}$$

or with $\mathbf{g}_n = \nabla f(\mathbf{x}_n) = -\mathbf{r}_n$:

$$\lambda_n = -\frac{\mathbf{r}_n^T \mathbf{g}_n}{\mathbf{r}_n^T \boldsymbol{A}\mathbf{r}_n}. \tag{2.13}$$

Finally, a simple recursion relation can be derived for successive gradients:

$$\begin{aligned}\mathbf{g}_{n+1} &= \nabla f(\mathbf{x}_{n+1}) = \boldsymbol{A}\mathbf{x}_{n+1} - \mathbf{b} = \underbrace{\boldsymbol{A}\mathbf{x}_n - \mathbf{b}}_{\mathbf{g}_n} + \lambda_n \boldsymbol{A}\,\mathbf{r}_n \\ \mathbf{g}_{n+1} &= \mathbf{g}_n + \lambda_n \boldsymbol{A}\,\mathbf{r}_n. \end{aligned} \tag{2.14}$$

In summary, the SD algorithm is given by Algorithm 6.

Algorithm 6 Steepest descent

Choose a suitable initial vector $\mathbf{x}_0$
$\mathbf{g}_0 = \boldsymbol{A}\mathbf{x}_0 - \mathbf{b}$
for $n = 0$ to $n_{\max}$ **do**
  $\mathbf{r}_n = -\mathbf{g}_n$
  $\lambda_n = \dfrac{\mathbf{r}_n^T \mathbf{r}_n}{\mathbf{r}_n^T \boldsymbol{A}\mathbf{r}_n}$
  $\mathbf{x}_{n+1} = \mathbf{x}_n + \lambda_n \mathbf{r}_n$
  if converged **then** EXIT
  $\mathbf{g}_{n+1} = \mathbf{g}_n + \lambda_n \boldsymbol{A}\mathbf{r}_n$
end for

We will illustrate the SD method guided by the following simple example

$$A = \begin{pmatrix} 0.001 & 0 \\ 0 & 0.01 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 0.001 \\ 0.002 \end{pmatrix}, \quad c = 0.0007. \tag{2.15}$$

The trajectories are plotted in Fig. 2.2 on different scales. The coordinates of the minimum are $\xi = (1, 0.2)$. A 6 digit accuracy takes 76 iterations. We see the oscillatory trajectory on all scales (panels) in figure 2.2. This is in strong contradiction to our general consideration that a good gradient based method ought to yield the exact solution in $\mathcal{O}(N)$ steps. The shortcoming is the orthogonality of the search direction. If the first gradient forms an angle of 45 degrees with the x_1-axis so do all following directions as they are orthogonal to each other. Due to the shape of the contour lines (metric) this is a particularly bad situation since the line minimum is always close to the x-axis. For spherical contour lines, i.e.: equal eigenvalues for all principle axes of the matrix $\boldsymbol{A}$ the iteration would reach the exact solution within two steps. In order to overcome this problem, more general search directions are required which account for the non-spherical metric. Such directions are called *conjugate directions*.

Conjugate Directions

Steepest Descent often finds itself taking steps in the same direction as earlier steps. The idea is now to pick a set of orthogonal *search directions* $\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{N-1}$. In each search direction we will take exactly one step and

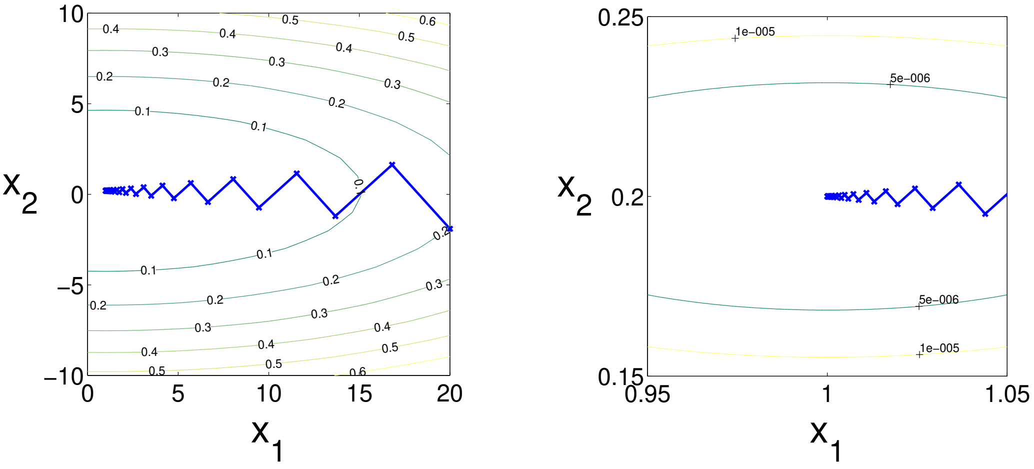


Figure 2.2: Steepest descent trajectory with initial point $(20, -1.9)$ for different resolutions.

that step will be just of the right length to size up evenly with $\mathbf{x}$. After N steps we will be done.

We choose for each step $n+1$ a point

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \lambda_n \mathbf{d}_n. \tag{2.16}$$

To find the value of λ_n we use the fact that the deviation $\mathbf{e}_{n+1}$ (the error vector) from the exact solution $\mathbf{x}$ should be orthogonal to $\mathbf{d}_n$, so that we never step into the direction $\mathbf{d}_n$ again. Using this, we have:

$$\mathbf{d}_n^T \mathbf{e}_{n+1} = 0. \tag{2.17}$$

Eq. (2.16) also gives

$$\mathbf{x} + \mathbf{e}_{n+1} = \mathbf{x} + \mathbf{e}_n + \lambda_n \mathbf{d}_n \tag{2.18}$$

which turns Eq. (2.17) into

$$\mathbf{d}_n^T (\mathbf{e}_n + \lambda_n \mathbf{d}_n) = 0,$$

with the result

$$\lambda_n = -\frac{\mathbf{d}_n^T \mathbf{e}_n}{\mathbf{d}_n^T \mathbf{d}_n}. \tag{2.19}$$

Unfortunately, nothing has been accomplished because λ_n cannot be calculated without knowing the $\mathbf{e}_n$; but if we knew $\mathbf{e}_n$ the problem would already have been solved.

The solution is to make the search directions $\boldsymbol{A}$-*orthogonal* or *conjugate* instead of orthogonal.

Theorem 2.1 *Two vectors $\mathbf{d}_i$ and $\mathbf{d}_j$ are conjugate if*

$$\mathbf{d}_i^T \boldsymbol{A} \mathbf{d}_j = 0. \tag{2.20}$$

We now require $\mathbf{e}_{n+1}$ to be conjugate to $\mathbf{d}_n$ and this condition is equivalent to finding the minimum point along the search direction $\mathbf{d}_n$:

$$\begin{aligned}
\frac{d}{d\lambda} f(\mathbf{x}_{n+1}) &= 0 \\
\nabla f(\mathbf{x}_{n+1})^T \frac{d}{d\lambda} \mathbf{x}_{n+1} &= 0 \\
-\mathbf{r}_{n+1}^T \mathbf{d}_n &= 0 \\
\mathbf{d}_n^T \left(\underbrace{\mathbf{b} - \boldsymbol{A}\mathbf{x}}_{=\mathbf{0}} - \boldsymbol{A}\mathbf{e}_{n+1} \right) &= 0 \\
\mathbf{d}_n^T \boldsymbol{A}\mathbf{e}_{n+1} &= 0.
\end{aligned}$$

As a byproduct of this calculation we find for the residue

$$\mathbf{r}_n = \mathbf{b} - \boldsymbol{A}\mathbf{x} - \boldsymbol{A}\mathbf{e}_n = -\boldsymbol{A}\mathbf{e}_n. \tag{2.21}$$

Following the derivation of Eq. (2.19) results in an expression for λ_n if the search directions are conjugate:

$$\lambda_n = -\frac{\mathbf{d}_n^T \boldsymbol{A}\mathbf{e}_n}{\mathbf{d}_n^T \boldsymbol{A}\mathbf{d}_n} \tag{2.22}$$

$$\overset{(2.21)}{=} \frac{\mathbf{d}_n^T \mathbf{r}_n}{\mathbf{d}_n^T \boldsymbol{A}\mathbf{d}_n}. \tag{2.23}$$

Unlike Eq. (2.19) we can calculate this expression. If we use for the search vector $\mathbf{d}_n$ the residue $\mathbf{r}_n$ then we recover the formula (2.12) for Steepest Descent.

To prove that this procedure really does compute $\mathbf{x}$ in N steps, we express the initial error term as a linear combination of search directions:

$$\mathbf{e}_0 = \sum_{j=0}^{N-1} \delta_j \mathbf{d}_j. \tag{2.24}$$

The search directions are all conjugate and this makes it possible to eliminate all δ_j values but one from Eq. (2.24) by multiplying it with $\mathbf{d}_k^T \boldsymbol{A}$:

$$\begin{aligned}
\mathbf{d}_k^T \boldsymbol{A}\mathbf{e}_0 &= \sum_j \delta_j \mathbf{d}_k^T \boldsymbol{A}\mathbf{d}_j \\
&\overset{(2.20)}{=} \delta_k \mathbf{d}_k^T \boldsymbol{A}\mathbf{d}_k.
\end{aligned}$$

Thus, we get

$$\delta_k = \frac{\mathbf{d}_k^T \boldsymbol{A} \mathbf{e}_0}{\mathbf{d}_k^T \boldsymbol{A} \mathbf{d}_k} \overset{(2.18)}{=} \frac{\mathbf{d}_k^T \boldsymbol{A} \left(\mathbf{e}_0 + \sum_{i=0}^{k-1} \lambda_i \mathbf{d}_i \right)}{\mathbf{d}_k^T \boldsymbol{A} \mathbf{d}_k} \overset{(2.16)}{=} \frac{\mathbf{d}_k^T \boldsymbol{A} \mathbf{e}_k}{\mathbf{d}_k^T \boldsymbol{A} \mathbf{d}_k} \tag{2.25}$$

By Eqs. (2.22) and (2.25), we find $\lambda_i = -\delta_i$ and this opens a new perspective for the error term:

$$\mathbf{e}_i = \mathbf{e}_0 + \sum_{j=0}^{i-1} \lambda_j \mathbf{d}_j = \sum_{j=0}^{N-1} \delta_j \mathbf{d}_j - \sum_{j=0}^{i-1} \delta_j \mathbf{d}_j = \sum_{j=i}^{N-1} \delta_j \mathbf{d}_j. \tag{2.26}$$

After N iterations, every component is cut away and $\mathbf{e}_N = \mathbf{0}$; we have convergence in N steps. Furthermore, checking on $\mathbf{e}$ during each step allows to decide to leave the algorithm prematurely if a certain accuracy has been reached.

Finally, we multiply Eq. (2.26) with $-\mathbf{d}_i^T \boldsymbol{A}$ and get:

$$-\mathbf{d}_i^T \boldsymbol{A} \mathbf{e}_j = -\sum_{k=j}^{N-1} \delta_i \boldsymbol{A} \mathbf{d}_k$$

with the result [using Eq. (2.21)]

$$\mathbf{d}_i^T \mathbf{r}_j = 0, \quad i < j, \tag{2.27}$$

because the $\mathbf{d}$-vectors are conjugate and the residual is evermore orthogonal to all old search directions.

Gram-Schmidt Conjugation

All that is needed now is a set of $\boldsymbol{A}$-orthogonal search directions $\{\mathbf{d}_i\}$ with $i = 0, \ldots, N-1$ and there is a simple way to generate them: The *conjugate Gram-Schmidt process*.

Suppose we have a set of N linearly independent vectors $\{\mathbf{u}_i\}$ with $i = 0, \ldots, N-1$, for instance unity vectors in the coordinate directions.

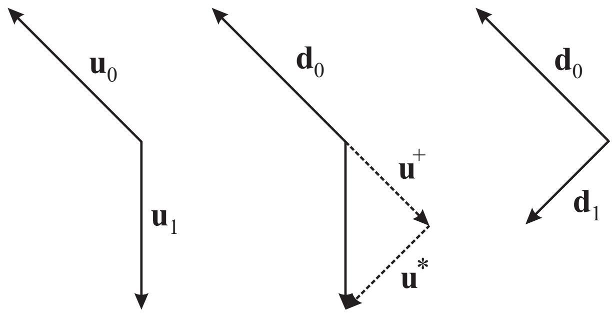


Figure 2.3: Gram-Schmidt conjugation of two vectors. Begin with two linearly independent vectors $\mathbf{u}_0$ and $\mathbf{u}_1$. Set $\mathbf{d}_0 = \mathbf{u}_0$. The vector $\mathbf{u}_1$ is composed of two components: $\mathbf{u}^*$ which is $\boldsymbol{A}$-orthogonal (or conjugate) to $\mathbf{d}_0$, and $\mathbf{u}^+$ which is parallel to $\mathbf{d}_0$. After conjugation, only the $\boldsymbol{A}$-orthogonal portion remains, and $\mathbf{d}_1 = \mathbf{u}^*$.

To construct $\mathbf{d}_i$ take $\mathbf{u}_i$ and subtract out all components that are not $\boldsymbol{A}$-orthogonal to the previous $\mathbf{d}$-vectors, as is demonstrated in Fig. 2.3. Thus, we set $\mathbf{d}_0 = \mathbf{u}_0$ and for $i > 0$ we choose

$$\mathbf{d}_i = \mathbf{u}_i + \sum_{k=0}^{i-1} \beta_{ik} \mathbf{d}_k, \tag{2.28}$$

with the β_{ik} defined for $i > k$. To find their values we use the same trick used to find the δ_i in Eq. (2.25):

$$\begin{aligned} \mathbf{d}_i^T \boldsymbol{A} \mathbf{d}_j &= \mathbf{u}_i^T \boldsymbol{A} \mathbf{d}_j + \sum_{k=0}^{i-1} \beta_{ik} \mathbf{d}_k^T \boldsymbol{A} \mathbf{d}_j \\ 0 &= \mathbf{u}_i^T \boldsymbol{A} \mathbf{d}_j + \beta_{ij} \mathbf{d}_j^T \boldsymbol{A} \mathbf{d}_j, \quad i > j \\ \beta_{ij} &= -\frac{\mathbf{u}_i^T \boldsymbol{A} \mathbf{d}_j}{\mathbf{d}_j^T \boldsymbol{A} \mathbf{d}_j}. \end{aligned} \tag{2.29}$$

The difficulty in using this method is that, obviously, all the old search vectors must be kept to construct the new search vector. Furthermore, $\mathcal{O}(N^3)$ operations are required to generate the full set.

Conjugate Gradients (CG)

In this method, the search directions are constructed by conjugation of the residuals, i.e.: we set $\mathbf{u}_i = \mathbf{r}_i$ and use the fact that the residual is orthogonal to all previous search directions as was demonstrated in deriving Eq. (2.27). Moreover, as each residual is orthogonal to the previous search directions, it is also orthogonal to the previous residuals, and we get

$$\begin{aligned} \mathbf{d}_i^T \mathbf{r}_j &\overset{(2.28)}{=} \mathbf{u}_i^T \mathbf{r}_j + \sum_{k=0}^{i-1} \beta_{ik} \mathbf{d}_k^T \mathbf{r}_j \\ 0 &\overset{(2.27)}{=} \mathbf{u}_i^T \mathbf{r}_j \end{aligned} \tag{2.30a}$$

and

$$\mathbf{d}_i^T \mathbf{r}_i = \mathbf{u}_i^T \mathbf{r}_i. \tag{2.30b}$$

Finally, our particular choice for the set $\{\mathbf{u}_i\}$ results in:

$$\mathbf{r}_i^T \mathbf{r}_j = 0, \quad i \neq j. \tag{2.30c}$$

We can also make use of Eq. (2.14) and $\mathbf{g}_n = -\mathbf{r}_n$ to find the recursion relation to be used to calculate the residual of step $n+1$ given the residual of step n:

$$\mathbf{r}_{n+1} = \mathbf{r}_n - \lambda_n \boldsymbol{A} \mathbf{d}_n.$$

After all these preliminaries we are now in a position to calculate the Gram-Schmidt coefficients for our particular case:

$$\begin{aligned} \mathbf{r}_i^T \mathbf{r}_{j+1} &= \mathbf{r}_i^T \mathbf{r}_j - \lambda_j \mathbf{r}_i^T \boldsymbol{A} \mathbf{d}_j \\ \lambda_j \mathbf{r}^T \boldsymbol{A} \mathbf{d}_j &= \mathbf{r}_i^T \mathbf{r}_i^T \mathbf{r}_{j+1} \\ \mathbf{r}_i^T \boldsymbol{A} \mathbf{d}_j &= \begin{cases} \frac{1}{\lambda_i} \mathbf{r}_i^T \mathbf{r}_i, & i = j \\ -\frac{1}{\lambda_{i-1}} \mathbf{r}_i^T \mathbf{r}_i, & i = j+1 \\ 0, & \text{otherwise}, \end{cases} \end{aligned}$$

and according to Eq. (2.29):

$$\beta_{ij} = \begin{cases} \frac{1}{\lambda_{i-1}} \frac{\mathbf{r}_i^T \mathbf{r}_i}{\mathbf{d}_{i-1}^T \boldsymbol{A} \mathbf{d}_{i-1}}, & i = j+1 \\ 0, & i > j+1. \end{cases}$$

Thus, most of the β_{ij} terms have disappeared. It is no longer necessary to store old search vectors to ensure $\boldsymbol{A}$-orthogonality. We now set, for simplification, $\beta_i = \beta_{i,i-1}$ and find:

$$\begin{aligned} \beta_i &\overset{(2.23)}{=} \frac{\mathbf{r}_i^T \mathbf{r}_i}{\mathbf{d}_{i-1}^T \mathbf{r}_{i-1}} \\ &\overset{(2.30b)}{=} \frac{\mathbf{r}_i^T \mathbf{r}_i}{\mathbf{r}_{i-1}^T \mathbf{r}_{i-1}}. \end{aligned}$$

Let's put it all together into one piece: After choosing a start vector $\mathbf{x}_0$ we get the first direction:

$$\mathbf{d}_0 = \mathbf{r}_0 = \mathbf{b} - \boldsymbol{A} \mathbf{x}_0.$$

From results calculated in step n we get the direction $\mathbf{d}_{n+1}$ for the next step from:

$$\lambda_n = \frac{\mathbf{r}_n^T \mathbf{r}_n}{\mathbf{d}_n^T \boldsymbol{A} \mathbf{d}_n} \tag{2.31a}$$

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \lambda_n \mathbf{d}_n \tag{2.31b}$$

$$\mathbf{r}_{n+1} = \mathbf{r}_n - \lambda_n \boldsymbol{A} \mathbf{d}_n \tag{2.31c}$$

$$\beta_{n+1} = \frac{\mathbf{r}T_{n+1} \mathbf{r}_{n+1}}{\mathbf{r}_n^T \mathbf{r}_n} = \frac{\mathbf{g}_{n+1}^T \mathbf{g}_{n+1}}{\mathbf{g}_n^T \mathbf{g}_n} \tag{2.31d}$$

$$\mathbf{d}_{n+1} = \mathbf{r}_{n+1} + \beta_{n+1} \mathbf{d}_n \tag{2.31e}$$

$$= \mathbf{g}_{n+1} + \beta_{n+1} \mathbf{r}_n. \tag{2.31f}$$

The corresponding algorithm is presented in symbolic form as Algorithm 7.

Algorithm 7 Conjugate gradient method for quadratic functions

Choose a suitable initial vector $\mathbf{x}_0$
$\mathbf{r}_0 = \mathbf{g}_0 = \nabla f\big|_{\mathbf{x}_0} = \boldsymbol{A}\mathbf{x}_0 - \mathbf{b}$
for $n = 0$ to N **do**
  $\mathbf{a}_n = \boldsymbol{A}\mathbf{r}_n$
  $\lambda_n = -(\mathbf{g}_n^T \mathbf{g}_n)/(\mathbf{r}_n^T \mathbf{a}_n)$
  $\mathbf{x}_{n+1} = \mathbf{x}_n + \lambda_n \mathbf{r}_n$
  $\mathbf{g}_{n+1} = \mathbf{g}_n + \lambda_n \mathbf{a}_n$
  $\beta_{n+1} = (\mathbf{g}_{n+1}^T \mathbf{g}_{n+1})/(\mathbf{g}_n^T \mathbf{g}_n)$
  $\mathbf{r}_{n+1} = \mathbf{g}_{n+1} + \beta_{n+1} \mathbf{r}_n$
  if converged **then** EXIT
end for

2.3 Conjugate Gradient for General Functions

We are now prepared to apply the conjugate gradient idea to general functions $f(\mathbf{x})$. To this end we expand $f(\mathbf{x})$ in each iteration about the actual reference point $\mathbf{x}_n$ of the n^{th} step

$$f(\mathbf{x}) = f(\mathbf{x}_n) + \mathbf{b}_n^T(\mathbf{x} - \mathbf{x}_n) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_n)^T \boldsymbol{A}_n (\mathbf{x} - \mathbf{x}_n) \tag{2.32a}$$

$$\mathbf{b}_n = \nabla f(\mathbf{x})\big|_{\mathbf{x}_n} \tag{2.32b}$$

$$(\boldsymbol{A}_n)_{ll'} = \frac{\partial^2}{\partial x_l \partial x_{l'}} f(\mathbf{x})\big|_{\mathbf{x}_n}, \tag{2.32c}$$

where the matrix $\boldsymbol{A}_n$ is the Hessian. Since (2.32) is a quadratic form it can easily be cast into the form (2.8), on which CG for quadratic problems was based, with one significant modification though: The matrix (Hessian)

$\boldsymbol{A}_n$ changes from iteration to iteration as does the vector $\mathbf{b}_n$. The iteration scheme 7 is nonetheless valid. All formulas (2.31) are still valid with the only modification that matrix $\boldsymbol{A}$ corresponds to the Hessian $\boldsymbol{A}_n$ of the Taylor expansion about the reference point $\mathbf{x}_n$.

Consequently, Algorithm 7 is still valid with the modification that $\boldsymbol{A}$ has to be replaced in each iteration by the respective Hessian $\boldsymbol{A}_n$. This implies that we have to be able to compute the Hessian which only in rare cases is given analytically and the numeric computation is inefficient. Fortunately, $\boldsymbol{A}_n$ enters only in conjunction with the iteration scheme for the gradients and in the line minimization or rather the expression for λ_n. These steps can be modified. First of all, we replace the update rule for $\mathbf{g}_{n+1}$ by the definition

$$\mathbf{g}_{n+1} = \nabla f(\mathbf{x})\big|_{\mathbf{x}_{n+1}}.$$

We merely require the knowledge of the gradient instead of the Hessian. Secondly, the parameter λ_n is obtain more directly via

$$\min_{\lambda} \; f(\mathbf{x}_n + \lambda \mathbf{r}_n) \;\Rightarrow \lambda_n.$$

The Eqs. (2.19), (2.27), and (2.30c) are, however, no longer valid for $|i-j| > 1$, since the matrix $\boldsymbol{A}$ changes from iteration to iteration. If the conjugacy relation (2.19) were still be valid then the arguments given for quadratic problems would still hold and convergence is reached within at most N steps. This, of course, cannot hold for arbitrary non-quadratic functions.

The corresponding algorithm is presented in symbolic form as Algorithm 8.

Algorithm 8 Conjugate gradient method

Choose a suitable initial vector $\mathbf{x}_0$
$\mathbf{r}_0 = \mathbf{g}_0 = \nabla f\big|_{\mathbf{x}_0}$
for $n = 0$ to $n_{\max}$ **do**
  $\min_\lambda \; f(\mathbf{x}_n + \lambda \mathbf{r}_n) \quad \Rightarrow \lambda_n$
  $\mathbf{x}_{n+1} = \mathbf{x}_n + \lambda_n \mathbf{r}_n$
  $\mathbf{g}_{n+1} = \nabla f\big|_{\mathbf{x}_{n+1}}$
  $\beta_{n+1} = (\mathbf{g}_{n+1}^T \mathbf{g}_{n+1})/(\mathbf{g}_n^T \mathbf{g}_n)$
  $\mathbf{r}_{n+1} = \mathbf{g}_{n+1} + \beta_{n+1} \mathbf{r}_n$
  if converged **then** EXIT
end for

2.3.1 The Determinant of Mega-Matrices

In various areas of research determinants of mega-dimensional matrices are required. All standard approaches fail if the matrix dimension is of the order $10^6 \times 10^6$ or even bigger. We will see that CG can - with some tricks

- be expedient in this case because an *exact* calculation of the determinant is not the aim here.

We first need to cast the problem into a form suitable for CG. We rewrite the matrix of interest, say $\boldsymbol{B}$, into:

$$\boldsymbol{B} = \mathbf{1} + \boldsymbol{A}. \tag{2.33}$$

We then use the matrix identity[1]

$$\text{det}\,[\exp(\boldsymbol{B})] = \exp\,[\,\text{tr}\ \boldsymbol{B}]\,. \tag{2.34}$$

The matrix function $\exp(\boldsymbol{B})$ always converges. We now set $\boldsymbol{C} = \exp(\boldsymbol{B})$ and write

$$\boldsymbol{B} = \ln(\boldsymbol{C}).$$

The matrix function $\ln(\boldsymbol{C})$ is defined as the (convergent) Taylor's expansion of the logarithm.[2] $\ln(\boldsymbol{C})$, of course, is a matrix. Thus, Eq. (2.34) results in

$$\det\ \boldsymbol{C} = \exp\{\,\text{tr}\ln\ \boldsymbol{C}\}\,,$$

and the matrix identity

$$\ln(\det\ \boldsymbol{C}) = \ \text{tr}[\ln(\boldsymbol{C})] \tag{2.35}$$

follows as a consequence. This yields together with Eq. (2.33):

$$\ln\left[\det(\mathbf{1}+\boldsymbol{A})\right] = \ \text{tr}\left[\ln(\mathbf{1}+\boldsymbol{A})\right] = \ \text{tr}\ \left(\boldsymbol{A}\int_0^1 d\lambda\,\frac{\mathbf{1}}{\mathbf{1}+\lambda\boldsymbol{A}}\right). \tag{2.36}$$

The correctness of the last step is easily shown by means of the spectral representation which plays an important role in the definition of matrix functions. (Obviously, for the above expression to make sense, it is required that the matrix $\mathbf{1} + \lambda\boldsymbol{A}$ is not singular!) The integral is computed numerically by some standard scheme (for instance an Gauss-algorithm)

$$\ln[\det(\mathbf{1}+\boldsymbol{A})] = \sum_{i=1}^{M}\omega_i\ \ \text{tr}\ \left(\boldsymbol{A}\frac{\mathbf{1}}{\mathbf{1}+\lambda_i\boldsymbol{A}}\right). \tag{2.37}$$

Here, the λ_i are the abscissas and the ω_i are the respective weights. (For a Gauss-Algorithm, the λ_i are the zeros of Legendre polynomials and the ω_i are the respective weights.)

We have to compute the trace of $\boldsymbol{A}(\mathbf{1}+\lambda_i\boldsymbol{A})^{-1}$ without being forced to compute the matrix elements of the matrix $\boldsymbol{A}(\mathbf{1}+\lambda_i\boldsymbol{A})^{-1}$ which, of course,

[1] J. HILGERT and K.-H. NEEB, *Lie-Gruppen und Lie-Algebren*, Vieweg, Barunschweig (1991), p. 14.

[2] See for instance: F.R. GRANTMACHER, *Matrixrechnung*, Bd. I, Deutscher Verlag der Wissenschaften (1958), chapter: Matrizenfunktionen.

is not feasible. There is another trick: The trace of a matrix, say $\boldsymbol{M}$, can be estimated stochastically by

$$\text{tr}_M = \frac{1}{L}\sum_{l=1}^{L} (\mathbf{r}^{(l)})^T M \mathbf{r}^{(l)} \;, \tag{2.38}$$

with the random variable tr_M and L is the number of random vectors $\mathbf{r}^{(l)}$ the elements of which are uncorrelated random numbers of zero mean and unit variance. Thus, the elements of $\mathbf{r}^{(l)}$ have the properties

$$\left\langle r_i^{(l)} \right\rangle = 0, \quad \forall\, i \tag{2.39a}$$

$$\left\langle r_i^{(l)}\; r_j^{(l')} \right\rangle = \delta_{ij}\delta_{ll'}. \tag{2.39b}$$

The mean of the random variable tr_M is then determined from:

$$\begin{aligned}
\langle\, \text{tr}_M \rangle &= \left\langle \frac{1}{L}\sum_{l=1}^{L}\sum_{ij}\; r_i^{(l)} M_{ij} r_j^{(l)} \right\rangle \\
&= \frac{1}{L}\sum_{l=1}^{L}\sum_{ij}\; M_{ij}\; \left\langle r_i^{(l)} r_j^{(l)} \right\rangle \\
&\overset{(2.39b)}{=} \frac{1}{L}\sum_{l=1}^{L}\sum_{ij}\; M_{ij}\delta_{ij} \\
&= \; \text{tr}\boldsymbol{M}.
\end{aligned}$$

We will see in a later chapter on Monte Carlo techniques that the standard error can be estimated by

$$\text{tr}(\boldsymbol{M}) = \; \text{tr}_M \pm \frac{\sigma}{\sqrt{L}}, \tag{2.40a}$$

with

$$\sigma = \frac{1}{L}\sum_{l=1}^{L} \left[(\mathbf{r}^{(l)})^T \boldsymbol{M}\mathbf{r}^{(l)} - \overline{\text{tr}_M}\right]^2. \tag{2.40b}$$

It is noteworthy that in many applications the number L of random vectors required for a certain accuracy decreases with the dimension of the matrices and typically $\mathcal{O}(10)$ vectors are sufficient for one percent accuracy.

Eq. (2.37) reads now: The solution $\mathbf{y}^{(l)}$ of the set of algebraic equations is computed by CG and the final result reads

$$\ln[\det(\mathbf{1} + \boldsymbol{A})] = \frac{1}{L}\sum_{i=1}^{M}\omega_i \sum_{l=1}^{L} (\mathbf{r}^{(l)})^T \frac{\boldsymbol{A}}{\mathbf{1} + \lambda_i \boldsymbol{A}} \mathbf{r}^{(l)}. \tag{2.41}$$

With the step

$$(\mathbf{1} + \lambda_i \boldsymbol{A})\, \mathbf{y}^{(l)} = \mathbf{r}^{(l)} \tag{2.42}$$

the CG-method comes into play and is used to determine the vectors $\mathbf{y}^{(l)}$. We use Eq. (2.42) to give Eq. (2.41) its final form:

$$\ln[\det(\mathbf{1} + \boldsymbol{A})] = \frac{1}{L} \sum_{i=1}^{M} \omega_i \sum_{l=1}^{L} (\mathbf{r}^{(l)})^T \boldsymbol{A} \mathbf{y}^{(l)}. \tag{2.43}$$

This method has the following advantages:

- Only operations matrix $\boldsymbol{A}$ times vector required.
- The dependence on i is hidden in the $\mathbf{y}^{(l)}$
- Operation count $(I + 1) * N_{nz} * L * M$, where I stands for the average number of CG steps and N_{nz} for the number of non-zero elements in matrix $\boldsymbol{A}$.
  Both L and M are typically of the order $10 - 100$.
  Hence, the dependence on the matrix dimension N is much less than that of the standard approach.The solution $\mathbf{y}^{(l)}$ of the set of algebraic equations is computed by CG.

2.4 Stochastic Optimization/Minimization

Literature

- J. SCHNAKENBERG, *Algorithmen in der Quantentheorie und Statistik.* Verlag Zimmermann-Neufang 1994.
- G.S. FISHMAN, *Monte-Carlo*, Springer 1995.
- W. KINNEBROCK, *Optimierung mit genetischen und selektiven Algorithmen*, Oldenbourg Verlag 1994.
- S. GEMAN and D. GEMAN, IEEE Trans. Patt. Anal. Machine Intell., **6**, 721–741 (1984).
- SZU and R. HARTLEY, *Fast simulated annealing*, Physics Letters A **122**, 157 (1987).

Nonlinear problems often have more than one minimum. Steepest Descent and Conjugate Gradients methods always yield the local minimum in the vicinity of the initial point. If there are not too many minima, we can start SD and CG at different initial points (chosen at random) to obtain the global minimum. However, there are many problems not accessible to this procedure. For instance if the problem displays many minima of about the same depth, SD and CG with random start points are very inefficient. Examples that are not suited for SD-/CG- based methods are

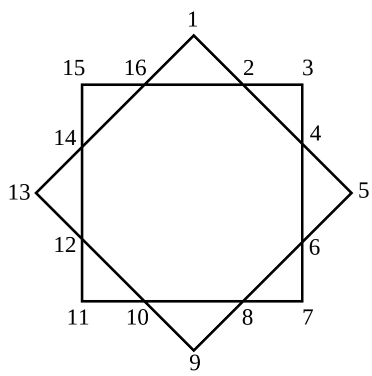
- The *Traveling Salesman* problem (TSP) is well known to quickly overcharge any computer if it is directly implemented. If the trip of the salesman consists of L cities, the number of possibilities is given by $L!$. In the TSP the salesman has to visit each city once and the length of the trip has to be minimized. The problem is of the form

$$f(\{i\}) = \sum_{\nu=1}^{L} \left|\mathbf{x}_{i_{\nu+1}} - \mathbf{x}_{i_\nu}\right|, \quad i_{L+1} = i_1.$$

  Here $\{i\}$ symbolizes a list of indices indicating the order in which the cities are visited. The minimum of the total cost function $f(\{i\})$ is required with $\left|\mathbf{x}_{i_{\nu+1}} - \mathbf{x}_{i_\nu}\right|$ the distance between two consecutive cities within the list $\{i\}$. $i_{L+1} = i_1$ indicates periodic boundary conditions.

  More realistic applications contain additional constraints and terms in the cost function.

- Redistribute the integer numbers from $1 - 16$ in a way that the sums along all edges have the same value. There are $16!$ possible index arrangements.



- Time tables at school

- Staff at an airport

2.5 Classical Simulated Annealing (CSA)

Literature:

- S. KIRKPATRICK, C.D. GELLAT, JR., and M.P. VECCHI, *Simulated Annealing*, Science **220**, 671 (1983).

The idea behind Simulated Annealing is borrowed from physics, specifically from thermodynamics. The method models the way liquids freeze and crystallize during the annealing process. At high temperatures the particles can move freely. Differences of potential energies of various configurations are overcome by the particles due to their high kinetic energy. When the liquid is cooled down slowly, an ordering process sets in, thermal mobility is lost and some optimal configuration (e.g. an ideal crystal) is achieved. This is a configuration where the cost function (free energy)

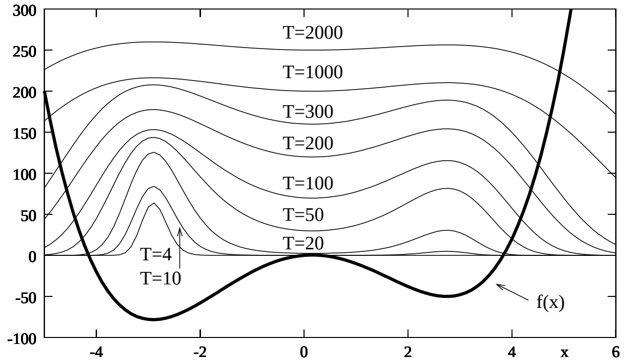


Figure 2.4: Cost function $f(x)$ (thick solid line) and acceptance probability $p_E(x|T)$ (thin solid lines) for various temperatures T.

attains its absolute minimum. However, if the liquid is cooled down too quickly, no ideal crystal is obtained and the systems ends in some local minimum of the energy (meta stable). This corresponds to a poly crystalline or amorphous state. As we are interested in finding global minima, we try to simulate the slow cooling process on the computer. To this end we introduce an *artificial temperature* T and a *probability distribution* $p_E(\mathbf{x}|T)$, the so-called *acceptance distribution* for a given configuration x and a given temperature T. The cost function to be minimized is denoted by $f(\mathbf{x})$ and can depend on a set of variables x that is either discrete or continuous. We set

$$p_E(\mathbf{x}|T) = \frac{1}{Z} e^{-[f(\mathbf{x})-f_{min}]/T}, \tag{2.44}$$

which corresponds to a BOLTZMANN distribution. Here Z is a normalization factor which corresponds to the partition function of the canonical ensemble. Notice that the knowledge of f_{min} is not really required as this factor can be absorbed in Z. Figure 2.4 shows the cost function (thick solid line)

$$f(x) = x^4 - 16x^2 + 5x \;, \tag{2.45}$$

which is frequently used to test optimization schemes. Its extrema have the coordinates:

x	f(x)
-2.9035	-78.3323
0.1567	0.3912
2.7468	-50.0589

Figure 2.4 also depicts the acceptance probability (thin solid lines) for various temperatures. At temperatures, much higher than the potential barrier $T \gg 50$, the Boltzmann distribution is rather flat. As soon as the temperature is lower than the potential barrier, e.g. at $T = 20$, the probability outside the double-well is almost zero and two separate peaks develop. If the temperature is even lower than the difference of the two minima, e.g.

at $T = 10$, the probability in the local minimum becomes negligible and the probability is concentrated merely around the global minimum.

It is the purpose of the distribution p_E to assign a high probability to states $\mathbf{x}$, where $f(\mathbf{x})$ is small, *i.e.:* states $\mathbf{x}$ close to the global minimum. A *random walker* will spend a long time near the optimum states. The significant difference to SD and CG lays in the possibility for the walker also to go *uphill* and to leave a local minimum. This possibility however depends on temperature. We define the expectation value of $f(\mathbf{x})$ at a given temperature T by

$$\langle f \rangle_T = \sum\!\!\!\!\!\!\int_{\mathbf{x}} p_E(\mathbf{x}|T)\, f(\mathbf{x}). \tag{2.46}$$

The minimum of $f(\mathbf{x})$ as well as the corresponding state $\mathbf{x}$ are found by cooling

$$\min_{\mathbf{x}} f(\mathbf{x}) = \lim_{T \to 0} \langle f \rangle_T \tag{2.47}$$

$$\mathbf{x}_{min} = \lim_{T \to 0} \max_{\mathbf{x}} p_E(\mathbf{x}|T). \tag{2.48}$$

Thus, an implementation of the CSA method requires the following three steps

1. **Generation of states**
2. **Acceptance of states**
3. **A cooling scheme**

ad 1: Generation of states In the vicinity of the current point $\mathbf{x}_n$ (n is the step index) we create a trial point $\mathbf{x}_t$ at random. The probability distribution of $\mathbf{x}_t$ for continuous problems is typically given by a Gaussian

$$p_x(\mathbf{x}_t - \mathbf{x}_n) \propto \prod_i e^{-(\mathbf{x}_t - \mathbf{x}_n)_i^2/(2\sigma^2)} \tag{2.49}$$

of a certain variance σ. If the curvature in various directions differs significantly, this form may be ill adapted. In such a case, it is expedient to use a p_x which employs the covariance matrix $\boldsymbol{C}$ instead of the variance σ:

$$p_x(\mathbf{x}_t - \mathbf{x}_n) \propto \exp\left\{\frac{1}{2}(\mathbf{x}_t - \mathbf{x}_n)\boldsymbol{C}^{-1}(\mathbf{x}_t - \mathbf{x}_n)\right\}.$$

For the Traveling Salesman, the tour is represented, e.g. by a list of indices $\{i_1, i_2, \ldots, i_L\}$ indicating the order in which the cities are visited. The so-called *lin-2-opt* move consists in exchanging randomly two indices in the list

$$\{i_1, \ldots i_{\nu-1}, i_\nu, \ldots i_\mu, i_{\mu+1} \ldots i_L\} \longrightarrow \{i_1, \ldots i_{\nu-1}, i_\mu, \ldots i_\nu, i_{\mu+1} \ldots i_L\}. \tag{2.50}$$

The advantage of *local moves* is that the change in cost function can usually be computed much faster than the total cost function

$$f(\{i\}) = \sum_{\nu=1}^{L} \left|\mathbf{x}_{i_{\nu+1}} - \mathbf{x}_{i_\nu}\right| , \tag{2.51}$$

with the periodic boundary condition $i_{L+1} = i_1$.

ad 2: Acceptance of states For the chosen $\mathbf{x}_t$ we define an *acceptance probability*

$$q = \min\left\{1, \frac{p_E(\mathbf{x}_t|T)}{p_E(\mathbf{x}_n|T)}\right\} , \tag{2.52}$$

which governs the acceptance of the new position $\mathbf{x}_t$: If $q = 1$ we always accept $\mathbf{x}_t$ as the new position $\mathbf{x}_{n+1}$. If $q < 1$ (*i.e.:* $\mathbf{x}_t$ is a worse choice than $\mathbf{x}_n$) we sample a pseudo random number r from a uniform distribution in the interval $[0, 1)$. (We call r an *auxiliary probability*.) If $q \geq r$ we accept $\mathbf{x}_t$ as the new state $\mathbf{x}_{n+1}$ in the next step. Otherwise the old state is kept $\mathbf{x}_{n+1} = \mathbf{x}_n$. By this trick it will be possible to leave a local minimum.

In this way we generate a series of points $\{\mathbf{x}_n\}$. These states $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_n)$ form a *Markov chain*. This means that

$$P(\mathbf{x}_{n+1}|\, \mathbf{x}_n, \mathbf{x}_{n-1} \ldots \mathbf{x}_1) = P(\mathbf{x}_{n+1}|\, \mathbf{x}_n) , \tag{2.53}$$

i.e.: the probability to pass to $\mathbf{x}_{n+1}$ depends only on the current point $\mathbf{x}_n$ and not on the history of the walker. It can be shown that the trajectory of the random walker reproduces the probability distribution

$$p_E(\mathbf{x}|T) = \lim_{L\to\infty} \frac{1}{L} \sum_{i=1}^{L} \delta(\mathbf{x} - \mathbf{x}_i). \tag{2.54}$$

This implies that expectation values at a given temperature T can be calculated via

$$\begin{aligned} \langle f(\mathbf{x})\rangle_T &= \sum\!\!\!\!\!\!\int_{\mathbf{x}} f(\mathbf{x})\, p_E(\mathbf{x}, T) \\ &= \lim_{L\to\infty} \frac{1}{L} \sum_{i=1}^{L} \int d^3x\, f(\mathbf{x})\, \delta(\mathbf{x} - \mathbf{x}_i) \\ &= \lim_{L\to\infty} \frac{1}{L} \sum_{i=1}^{L} f(\mathbf{x}_i) \end{aligned} \tag{2.55}$$

by averaging over the Markov chain.

Real problems often display several different scales. Consider as an example the traveling salesman at his/her trip from town to town in Germany. The distance from one town to another strongly depends on whether the town is in the same agglomeration or not. Thus we have two different length scales:

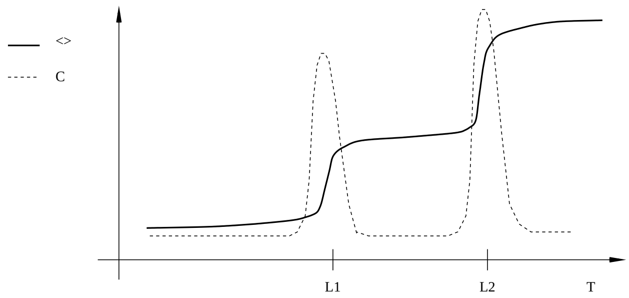


Figure 2.5: Cost function (solid line) and specific heat (dotted line) of the traveling salesman problem with two length scales L_1 and L_2

- Distance between towns of the same agglomeration.
- Distance between different agglomerations.

The existence of different scales is reflected by the way energy decreases when we lower the temperature. At extremely high temperatures agglomeration does not play a role. The salesman travels at random. In the cooling procedure first the order of the agglomerations to be visited is decided on. Only then the trip inside each agglomerations is fixed.

We introduce now the difference in 'energy' of the two configurations $\{i\}$ and $\{i'\}$ [see Eq. (2.50)] as the difference in the corresponding cost functions (2.51):

$$E(\{i'\}) - E(\{i\}) = \overline{\mathbf{x}_{i_{\nu-1}}\mathbf{x}_{i_\mu}} + \overline{\mathbf{x}_{i_\nu}\mathbf{x}_{i_{\mu+1}}} - \overline{\mathbf{x}_{i_{\nu-1}}\mathbf{x}_{i_\nu}} - \overline{\mathbf{x}_{i_\mu}\mathbf{x}_{i_{\mu+1}}},$$

with $\overline{\mathbf{x}_{i_\nu}\mathbf{x}_{i_\mu}} = \left|\mathbf{x}_{i_\nu} - \mathbf{x}_{i_\mu}\right|$ the distance between the cities ν and μ in configuration $\{i\}$. During the simulation for one specific temperature T we calculate

$$\langle E\rangle = \frac{1}{L}\sum_{\{i\}} E(\{i\}), \quad \text{and} \quad \langle E^2\rangle = \frac{1}{L}\sum_{\{i\}} E^2(\{i\}),$$

with the variance

$$\langle \Delta E^2\rangle = \quad \langle E^2\rangle - \langle E\rangle^2.$$

Here, $\sum_{\{i\}}$ indicates the sum over configurations covered in this simulation. This allows to introduce a *specific heat* C defined as

$$C \stackrel{\text{def}}{=} \frac{\partial \langle E\rangle}{\partial T} = \frac{\langle \Delta E^2\rangle}{T^2}. \tag{2.56}$$

Considering Fig. 2.5 we get a first cooling rule: Away from phase transitions we can cool down quickly. However, we have to be careful in the vicinity of such phase transitions, and the specific heat C with its rapid variation around phase transitions is a good indicator of critical regions.

Alternative Proposals

- *Threshold Acceptance:* A new configuration $\mathbf{x}_t$ is generated and this new configuration can, of course, be worse than the one used in the previous step, namely $\mathbf{x}_n$. Nevertheless, if the cost function $f(\mathbf{x}_t) < f(\mathbf{x}_n)+T$, with T some tolerance level (threshold), then $\mathbf{x}_t$ is accepted as a new position of the random walker $\mathbf{x}_{n+1}$. During the iteration the threshold is continuously reduced. This allows rather effectively to leave local minima.

- *Deluge Algorithms:* Accept new configurations $\mathbf{x}_t$ only if $f(\mathbf{x}_t) > T$ with T the acceptance level. T is continuously increased during the iterations; the landscape is 'flooded' until only the summits of the mountains, and finally only the summit of the biggest mountain is above the water level.

- *Genetic Algorithms:* Such algorithms will be presented in Sec. 2.8.

ad 3: Cooling Strategies An important first step is the choice of the initial temperature T_0. At this temperature it should be possible to cover the best part of the configuration space and it is a rule of thumb that at this temperature approximately 80% of the configurations should be accepted. This can easily be done by choosing an approximate value for T_0 and by performing N steps. N_r configurations have been rejected and if $N_r/N < 0.2$ T_0 is accepted as an initial temperature, otherwise we double T_0 and try again.

Nevertheless, a better estimate can be worked out. We start with Eq. (2.56) and find by integration

$$\langle E\rangle(T) - \langle E\rangle(\infty) = \int_{\infty}^{T} dT' \, \frac{\langle \Delta E^2\rangle(T')}{T'^2} \approx \frac{\langle \Delta E^2\rangle(\infty)}{T}, \tag{2.57}$$

with $\langle E\rangle(\infty)$ and $\langle \Delta E^2\rangle(\infty)$ the mean value and variance of the energy at $T = \infty$, respectively. We now choose T_0 in such a way that the energy $E(T_0)$ lies just within the area of fluctuation of $E(\infty)$, *i.e.:*

$$\langle E\rangle(T_0) = \langle E\rangle(\infty) - \sqrt{\langle \Delta E^2\rangle(\infty)}.$$

This gives, using Eq. (2.57), the initial temperature

$$T_0 = \sqrt{\langle \Delta E^2\rangle(\infty)}.$$

Thus, to find the best T_0 it is necessary to start a random walk of length N during which all configurations will be accepted and to measure $\sqrt{\langle \Delta E^2\rangle(\infty)}$ along this walk.

To proceed in the simulation we let the walker then do a fixed number of N steps at every fixed temperature T_k and then the temperature is lowered according to a simple formula. Of common use is the formula

$$T_k = T_0\, q^k \quad \text{with e.g. } q = 0.95, \tag{2.58}$$

which has the obvious drawback of not taking into account any phase transitions. Close to phase transitions fluctuations are rather large and the simulation could end up in a configuration far away from the optimum configuration.

Thus, a more gentle cooling strategy requires that the probability of acceptance for two consecutive temperatures T_k and T_{k+1} differ by a sufficiently small amount, *i.e.:*

$$\frac{1}{1+\delta} < \frac{p_E(\mathbf{x}|T_k)}{p_E(\mathbf{x}|T_{k+1})} < 1 + \delta, \tag{2.59}$$

with $\delta \ll 1$.

If we make use of the BOLTZMANN distribution (8.1) for equilibrium we get

$$p_E(\mathbf{x}|T) \propto \exp\left\{ -\frac{E(\{i\}) - E_{min}}{T} \right\},$$

with E_{min} the lowest energy. Because of this, the left hand side of the inequality (2.59) is always obeyed for $T_{k+1} < T_k$. Thus

$$\exp\left\{ -\frac{E(\{i\}) - E_{min}}{T_k} + \frac{E(\{i\}) - E_{min}}{T_{k+1}} \right\} < 1 + \delta$$

is additionally to be satisfied. This inequality can be solved for T_{k+1} and results in:

$$T_{k+1} > \frac{T_k}{1 + T_k \frac{\ln(1+\delta)}{E(\{i\}) - E_{min}}}. \tag{2.60}$$

If we wanted to satisfy this inequality for all configurations $\{i\}$ we would end up with $T_{k+1} = T_k$. A rather reliable estimate can be derived from

$$E(\{i\}) - E_{min} \approx 3\sqrt{\langle \Delta E^2 \rangle (T_k)}, \tag{2.61}$$

and this results in the cooling strategy

$$T_{k+1} > \frac{T_k}{1 + T_k \frac{\ln(1+\delta)}{3\sqrt{\langle \Delta E^2 \rangle (T_k)}}}. \tag{2.62}$$

Since we are interested in a cooling as fast as possible, we take the lowest possible Temperature T_{k+1} and therefore exchange the greater sign in (2.62) by an equal sign.

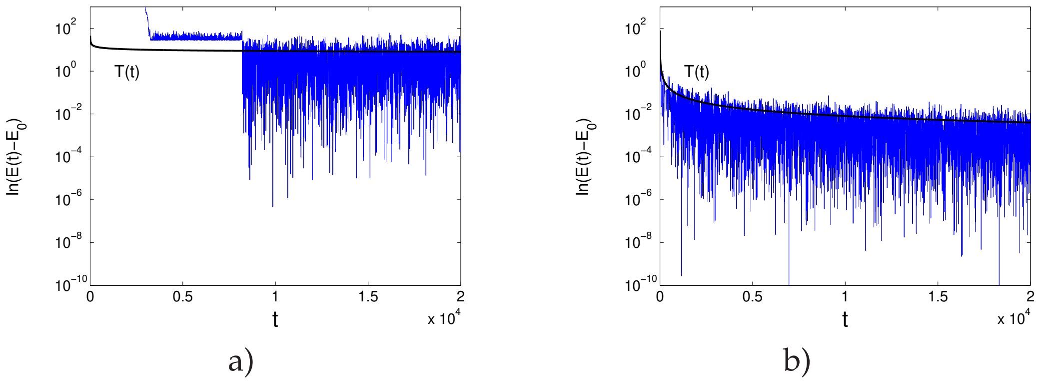


Figure 2.6: Illustration of a) CSA and b) FSA with parameters $T_0^E = 80$, the temperature of the BOLTZMANN engine. $T_0^x = 1$ corresponds to the $2\sigma^2$ of Eq. (2.61). The blue curve depicts the values of the logarithm of $E(t) - E_0$ in time t. E_0 is the value of the energy at the global minimum. The temperature $T(t)$ is chosen according to the optimal cooling schedule.

It has been shown by GEMAN and GEMAN that the optimal cooling schedule for the BOLTZMANN engine is given by

$$T(t) \sim \frac{1}{\ln t}, \tag{2.63}$$

with t a 'time' parameter, basically the step counter. It guarantees that the global minimum is visited with probability ONE. The convergence, however, is rather slow in view of the logarithmic cooling rate.

During the annealing process it is expedient to reduce the extent of the proposal in such a way that an acceptance rate of about 50% is ensured. Typically, we measure the acceptance rate during $N = 100$ steps. If the acceptance rate is below 40% we decrease the size of the moves, if it is above 60% we increase the size of the moves, otherwise the proposal distribution is acceptable.

We, finally, need a criterion which can be used to determine whether the simulation is converged or not, *i.e.:* is the current temperature T_k the end temperature T_e? To decide this we use the requirement that the expectation value $\langle E\rangle(T_e)$ should differ from the optimum energy E_{min} only by a sufficiently small parameter ε. This can be be expressed as:

$$\frac{\langle E\rangle(T_e) - E_{min}}{\langle E\rangle(T_0) - \langle E\rangle(T_e)} < \varepsilon. \tag{2.64}$$

This results in the criterion

$$\frac{\langle \Delta E^2\rangle(T_e)}{T_e\left[\langle E\rangle(T_0) - \langle E\rangle(T_e)\right]} < \varepsilon \tag{2.65}$$

which can be derived from Eq. (2.64) with the help of Eqs. (2.57) and (2.56).

Algorithm 9 CSA

Choose a suitable initial vector $\mathbf{x}_0$
$T_0 = T_0^E$
for $j = 0$ to $j_{\max}$ **do**
  for $n = 0$ to N **do**
    generate trial state $\mathbf{x}_t$ according to $p_x(\mathbf{x}_t - \mathbf{x}_n)$
    compute $q = \min(1, p_E(\mathbf{x}_t)/p_E(\mathbf{x}_n))$
    if $q = 1$; $\mathbf{x}_{n+1} = x_t$; **else**
      random number $r \in [0, 1)$
      if $q > r$; $\mathbf{x}_{n+1} = \mathbf{x}_t$; **else** $\mathbf{x}_{n+1} = \mathbf{x}_n$;
  end for
  determine T_{j+1} (e.g. $T_{j+1} = T_j * 0.99$)
  if converged **then** EXIT
end for

One shortcoming of the CSA method is, that due to the exponential character of the BOLTZMANN distribution, only *short moves*, or rather small modifications, are allowed. Therefore it takes a long time to escape from a local minimum. This behavior is illustrated in figure 2.6 for the polynomial cost function (2.45). Even for the simple double well potential we observe that the walker gets stuck in the local minimum for a long time before it eventually overcomes the potential barrier which is asymptotically guaranteed by the optimal cooling schedule. In this graph T_0^E is the temperature of the BOLTZMANN engine and T_0^x corresponds to the $2\sigma^2$ according to Eq. (2.61) which governs the step size in the stochastic minimization of a function. At the beginning T_0^E and T_0^x are chosen to result in acceptance of 80% of the trial value.

Improved algorithms should allow some longer ranging deviations for the Boltzmann distribution. Such a strategy is introduce in the next section.

2.6 Fast Simulated Annealing (FSA)

Instead of the BOLTZMANN function Fast Simulated Annealing uses the CAUCHY function for the acceptance distribution. Therefore it is called the CAUCHY *engine* in contrast to the BOLTZMANN engine. The D-dimensional CAUCHY distribution is given by

$$p_x(\Delta\mathbf{x}|T) = \frac{T}{\left[(\Delta\mathbf{x})^2 + T^2\right]^{\frac{D+1}{2}}}, \tag{2.66}$$

where D is the dimension of the configuration space. Due to its long ranging tails, the CAUCHY distribution has the advantage to allow occasionally larger changes in configuration space, while the short range moves are still Gaussian distributed with a variance $\sigma^2 = 2\,T^2/(D+1)$.

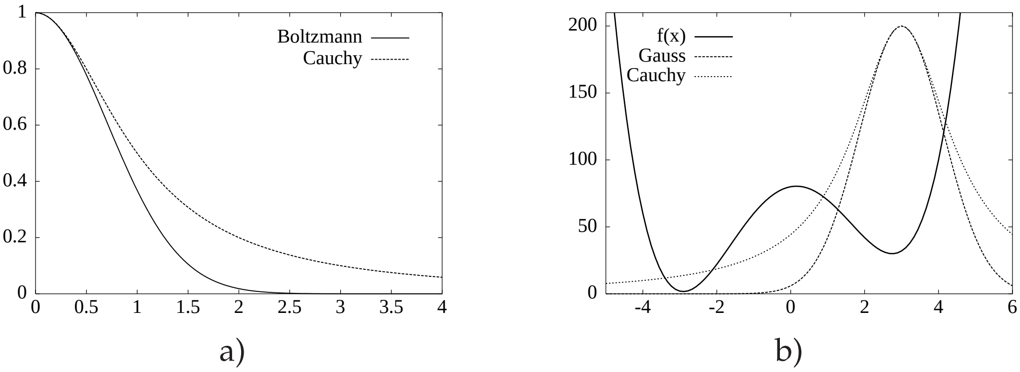


Figure 2.7: Comparison between a) BOLTZMANN (GAUSS) and CAUCHY distribution. Frame b) shows the finite probability of a random walker to jump from the local into the global minimum if it is controlled by a CAUCHY distribution.

SZU and HARTLEY derived that the optimal cooling schedule for the CAUCHY engine is given by

$$T(t) \sim \frac{1}{t}, \tag{2.67}$$

which is a considerable speed increase compared to the logarithmic cooling schedule of CSA.

In figure 2.8 CSA and FSA are compared based on the double well potential (2.45) discussed before. The temperature entering the proposal distribution is adjusted every 100 steps to ensure 50% acceptance rate. For each annealing step N, the lowest energy (value of the cost function) is stored in $E(N)$. The entire annealing procedure, covering $N_{\max} = 10000$ steps is repeated 1000 times and average $\langle E(N) \rangle$ is computed for each value of N separately. We see that FSA is indeed superior.

2.7 Generalized Simulated Annealing (GSA)

- C. TSALLIS and D.A. STARIOLO, *Generalized simulated annealing*, Physica A **233**, 395 (1996).
- I. ANDRICIOAEI and J.E. STRAUB, *Generalized simulated annealing algorithms using Tsallis statistics: Application to conformational optimization of a tetra-peptide*, Phy. Rev. E, **53**, 1996.

For the purpose of a further refinement it is convenient to introduce an extended notion of entropy depending on a parameter ε

$$S_\varepsilon = -\frac{1}{\varepsilon} \sum_{i=1}^{N} p_i \left(1 - p_i^{-\varepsilon}\right), \tag{2.68}$$

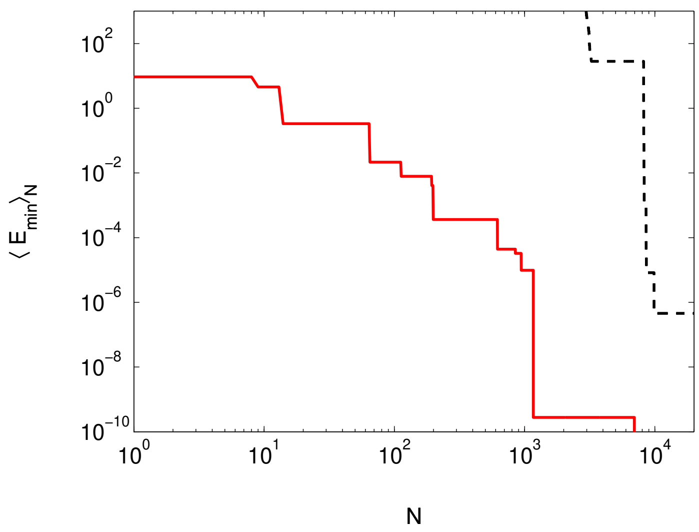


Figure 2.8: Average minimum energy $\langle E_{\min} \rangle_N$ reached in step N for FSA (solid red curve) and CSA (dashed black curve). Initial temperature is $T_0 = 1000$.

the TSALLIS entropy. The limit $\varepsilon \to 0$ the TSALLIS entropy equals the usual definition of entropy, *i.e.:*

$$\lim_{\varepsilon \to 0} S_\varepsilon = \lim_{\varepsilon \to 0} -\frac{1}{\varepsilon} \sum_{i=1}^{N} p_i \left(1 - e^{-\varepsilon \ln p_i}\right) = -\sum_{i=1}^{N} p_i \ln p_i \,. \tag{2.69}$$

Given the common axioms, (2.68) is the most general form of an entropy. It also governs the statistics on fractal structures.

Let us now consider a micro canonical ensemble. This means that the total internal energy is known and constant

$$E = \sum_{i=1}^{N} p_i \, E_i. \tag{2.70}$$

Maximizing the entropy for a given E yields the TSALLIS *distribution*

$$p_i = \frac{1}{Z}(1 + \beta\varepsilon E_i)^{-1/\varepsilon}, \tag{2.71}$$

with its limit $\varepsilon \to 0$

$$\lim_{\varepsilon \to 0} \frac{1}{Z} \, e^{-\frac{1}{\varepsilon}\ln(1+\beta\varepsilon E_i)} = \frac{1}{Z} e^{-\beta E_i}, \tag{2.72}$$

which, corresponds, again, to the classical distribution. As usual Z denotes the partition function and β is inversely proportional to the energy E:

$$Z = \sum_{i=1}^{N} (1 + \beta\varepsilon E_i)^{-1/\varepsilon}, \quad E = k_B \, T = \frac{1}{\beta}. \tag{2.73}$$

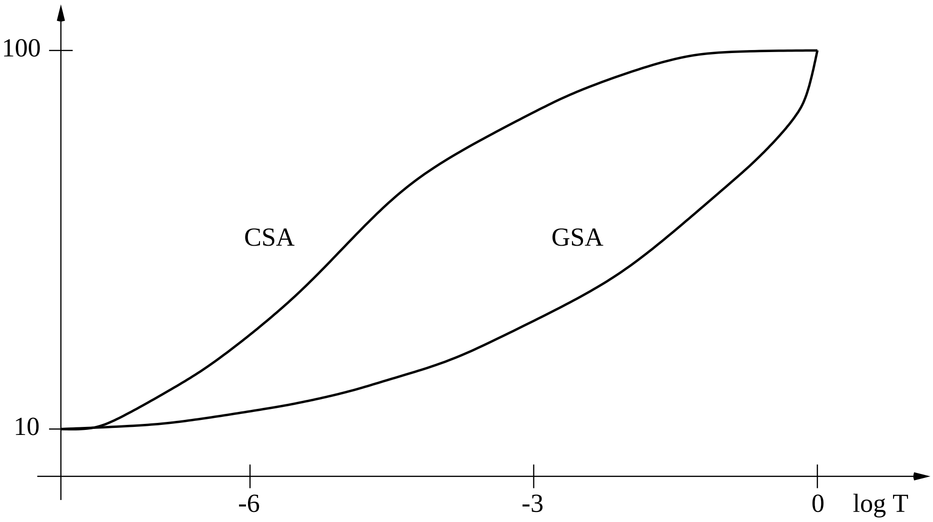


Figure 2.9: Comparison CSA and GSA for the 150 town problem. Convergence is much faster for the GSA method using the Tsallis entropy.

Instead of the BOLTZMANN distribution we can use the TSALLIS distribution to decide the acceptance of a step of the random walker. In doing so we gain an additional parameter ε that has to be adapted to the problem. An engine that

- chooses the moves according to a CAUCHY distribution
- decides upon the acceptance according to a TSALLIS distribution

is called *Generalized Simulated Annealing*. For some problems, it has the advantage of much faster convergence. This is illustrated in Figure 2.9 for a Traveling Salesman Problem with 150 towns.

2.8 Genetic algorithms

Such an algorithm does not only allow local modifications (called *mutations*) but also the possibility of *mating*. To explain this, let us assume that a configuration is mapped on to a string s (e.g.: a DNA, a chromosome, ...). Then mating describes the splitting of the strings of two individuals into two substrings $s_i^{(1)}, i = 1, 2$ and $s_i^{(2)}, i = 1, 2$, respectively, which will then be used to build two 'new' individuals, namely $\left\{s_1^{(1)}, s_2^{(2)}\right\}$ and $\left\{s_1^{(2)}, s_2^{(1)}\right\}$.

This idea will be illustrated using the TSP problem. The sequence of cities, as they appear in the tour, is no permissible string since mutation could lead to strings in which certain cities occur repeatedly while others are missing. A permissible string is the sequence of indices s $= \{s_1, \ldots, s_N\}$, where s_i indicates a city not yet visited.

A sample list for a 10-city problem is given, for instance, by

$$\mathrm{s} = \{9, 4, 3, 3, 5, 1, 4, 2, 2, 1\}.$$

The procedure is illustrated in Table 2.1.

Table 2.1: Sample TSP tour to illustrate the genetic algorithm.

s	1	2	3	4	5	6	7	8	9	10		tour
9	1	2	3	4	5	6	7	8	[9]	10	→	9
4	1	2	3	[4]	5	6	7	8	10		→	4
3	1	2	[3]	5	6	7	8	10			→	3
3	1	2	[5]	6	7	8	10				→	5
5	1	2	6	7	[8]	10					→	8
1	[1]	2	6	7	10						→	1
4	2	6	7	[10]							→	10
2	2	[6]	7								→	6
2	2	[7]									→	7
1	[2]										→	2

The first column contains the string s. We start out with the original list of cities in natural order $\{1, \ldots, 10\}$ as depicted in the first row. The column s indicates the contents of which column are to be transferred to the column named *tour*. Thus, the content of the first row, column 9 is bracketed to indicate that this content is to be moved to the column named *tour*. In this case, city 9 is moved to the corresponding element in column tour. Next the list of cities, without city 9, is compressed (city 9 is eliminated) and transferred into the second row. Column s indicates that the content of column 4 has to taken from the actual list. This element, marked in brackets, is city 4 and the index is transferred to column tour, and move on to the next row.

Eventually the tour represented in city indices reads according to column tour: tour $= \{9, 4, 3, 5, 8, 1, 10, 6, 7, 2\}$ as shown in Table 2.1.

The procedure is as follows

- We start with an ensemble of L individuals (states).
- For each individual we introduce a random local modification (mutation).
- Individuals are pairwise combined by the mating process resulting in $2L$ individuals.
- Half of the populations are eliminated. *I.e.:* the L individuals with the lowest value for the cost function survive.